Note on trees

Definition 1. A *tree* is a connected graph with no cycles.

A *forest* is a graph with no cycles (which is not necessarily connected).

From computer science, you might be familiar with what are called *rooted trees* such as recursion trees or binary search trees.

Definition 2. A *rooted tree* is digraph obtained from a tree T and a special vertex $r \in V(T)$ called the *root* by directing every edge "towards" the root (e.g., from the vertex farthest from the root to the vertex closest to the root).

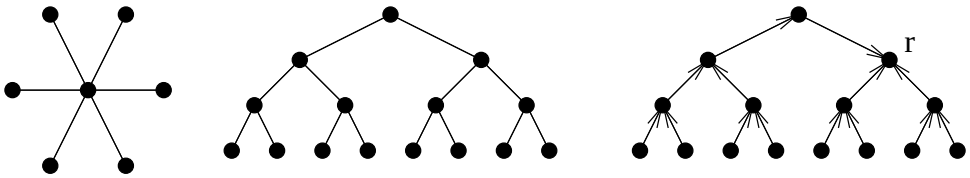

Figure 1: Some examples of trees. The tree on the right is the rooted tree obtained from the tree in the middle and r.

Lemma 1. *If T is a tree with at least 2 vertices then T has a vertex of degree 1.*

Proof. Suppose the lemma is false. Then there is a tree T with at least 2 vertices with no vertices of degree 1.

Since T is connected and has at least 2 vertices, T has no vertex of degree 0 (otherwise, there is no paths from a degree 0 vertex to any other vertex).

Therefore, T only has vertices of degree at least 2.

We will now find a cycle in T, which is a contradiction. We build a walk as follows.

- Start at any vertex $v \in V(T)$
- Move along unused edges until either
    1. we get stuck (because there are no more unused edges around the current vertex), or
    2. we visit a vertex we have previously visited.

We note that we cannot actually get stuck. Indeed any vertex which we visit for the first time has at least one unused edge incident to it (because it started with at least 2 unused edges).

Therefore, we stop when we re-visit an already visited vertex. But now the subpaths from that vertex to itself is a cycle. This is a contradiction to T being a tree (since trees have no cycles). □

The previous lemma allows us to prove the following interesting fact.

Theorem 1. *Every tree on n vertices has exactly $n-1$ edges.*

Proof. We prove this theorem by induction on n, the number of vertices of the tree.

If T has a single vertex then it has no edges so the theorem is verified for $n = 1$.

Now for the inductive step.

Suppose all trees on $n-1$ vertices have $n-2$ edges. We will show that an arbitrary tree T on n vertices has $n-1$ edges.

By the previous lemma, there is a vertex v of degree 0 in T.

We claim that $T - v$ is a tree (on $n - 1$ vertices). $T - v$ has no cycles as a cycle in $T - v$ is a cycle in T. $T - v$ is connected as a path in T between two vertices $u, w \in V(T - v)$ is still a path in $T - v$ since all internal vertices (non-endpoints) of a path have degree at least 2.

Therefore, by induction, $T - v$ has exactly $n - 2$ edges. T has exactly one more vertex than $T - v$ and exactly one more edge than $T - v$ since v has degree exactly 1. Therefore, T has $n - 1$ edges as required.

We have therefore proven the theorem by induction. □

Theorem 2. *Let $T = (V, E)$ be a tree and $e \notin E(T)$. Then the graph $(V, E \cup \{e\})$ contains a unique cycle C (and C contains e).*

The new graph $(V, E \cup \{e\})$ is sometimes denoted $T + e$.

Proof. Let $e = (u, v)$. Since T is connected, there is a path P_1 from u to v in T. This path is a cycle in $T + e$ (by our definition of cycle).

If there is another cycle in $T + e$, we can obtain (from it) other another path P_2 between u and v in T (check this from the definitions). But now, concatenating P_1 with P_2 backwards gives a walk from u to u in T. Since P_1 and P_2 are different, we can obtain a cycle from this walk (e.g., by looking at the first time P_2 reversed differs from P_1 and the first time the walk uses a vertex of P_1 again after that). This is a contradiction to T having no cycles. □

Minimum spanning tree

Suppose we want to build a network to connect n nodes (of, for example, computers). These nodes need not be connected directly to each other. Furthermore, we want to build the cheapest such networks.

Problem 1. Input: A connected graph $G = (V, E)$ and weights $w_e \geq 0$ for each edge $e \in E$ **Output:** A subset F of E such that (V, F) is connected and given these restrictions, $\sum_{e \in E} w_e$ is maximized.

Here, weights are thought of as costs for "building" an edges.

We can immediately sees that if (V, F) is a tree then we have satisfied the connectivity condition. Furthermore, if we have a cycle in (V, F), we can remove an edge of the cycle from F to get a cheaper network which is still connected. This is why this problems is called the *minimum spanning tree* problem.

By Theorem 1, we know that any solution will contain exactly $|V| - 1$ edges. In particular, we can pick any tree in the case where all edge have the same weight.

Algorithm 1. (Kruskal's algorithm)

Initialize F to the empty set.
Sort the edges in ascending order of weights
For each edge e in this ordering.
  If $(V, F \cup \{e\})$ does not contain a cycle then add e to F
Return F

Theorem 3. *Kruskal's algorithm returns a minimum spanning tree.*

Proof. To prove this theorem, we need to show that the set of edges F returned by Kruskal's algorithm has 3 properties.

(1) F is connected.

(2) F has no cycles.

(3) F has minimum weight.

First, we prove (1). Let u, v be two arbitrary vertices in V. We will show that there is a path between them. Let P be a path in G from u to v (P exists since G is connected). For each edge of this path $e \in E(P)$ that is not in F, there is a path P_e between its endpoints in F (otherwise, e should be added to F when

it was considered in the for-loop). Replacing each edge e with the path P_e gives a walk from u to v in T. From this walk, we can obtain a path from u to v.

Second, we prove (2). At the beginning of the algorithm F is empty and thus contains no cycles. If at some iteration i of the for loop, the current set of edges F_i yield no cycles (in (V, F_i)) then because of the condition we check, there is no cycle in the set F_{i+1} of the next iteration.

Finally, we prove (3). Let F^* be a minimum spanning tree which maximizes the number of edges of F it contains (that is, it maximizes $|F \cap F^*|$. If $F = F^*$ then we have proven (3) (since F^* is a minimum spanning tree).

Let F_i be the set of edges at the first iteration of the for-loop where we added an edge $e \notin F^*$ to F_i. So $F_{i+1} = F_i \cup \{e\}$. By Theorem 2, $(V, F^* \cup \{e\})$ contains a unique cycle C.

Since (V, F_{i+1}) contains no cycles, there is an edge f of C not in F_{i+1}. We claim that $w_f \geq w_e$.

Suppose not. If $w_f < w_e$ then we claim that our algorithm could have picked f instead of e (contradiction our choice of e). Indeed, if we cannot pick f, it is because f is contained in a cycle in $(V, F_i \cup \{f\})$ which is a path P between the endpoints of f in (V, F_i). Since F_i is contained in F^* (because e is the first edge not in F^*), P is a path in (V, F^*). But f is also in F^*. So P together with f is a cycle in F^*. Contradiction (to F^* being a tree).

Therefore, $w_f \geq w_e$. Since C is the unique cycle in $(V, F^* \cup \{e\})$ and f is in C, removing f from $(V, F^* \cup \{e\})$ gives a tree (which is still connected). But this new tree has lower or equal weight than (V, F^*) and contains more edges of F than F^* (namely, e). Contradiction to our choice of F^*. □

Note that for Kruskal's algorithm, we need to be able to determine if adding an edge e to an existing set of edges F creates a cycle in $(V, F \cup \{e\})$. This can be easily done by checking if the endpoints of e are in the same connected component in (V, F) (or, equivalently, if there is a path between the endpoint in (V, F)).

Checking if two vertices are in the same connected component is easy. For example, we can use depth-first search (DFS) (which you have seen in COMP 250) starting at one endpoint and see if we can reach the other.